Amiga® Mail

Volume II

May/June 1991

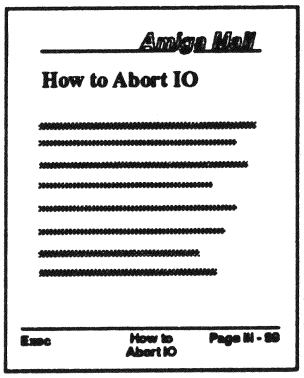
The Commodore Amiga Technical Newsletter

Copyright 1991 Commodore-Amiga, Inc. All rights reserved. Some of the materials used herein have been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions.

Amiga Mail
Volume II

Amiga Mail is published six times annually by the Commodore Applications and Technical Support Group (CATS). Its purpose is to support third-party developers of Amiga® software and hardware worldwide by providing technical information and programming guidelines. Each issue features a collection of news and technical articles of special interest to Amiga product developers. The technical articles in each issue can be saved for future use in the Amiga Mail Tech Notes binder, a reference notebook containing all the articles from past issues of Amiga Mail.

For filing technical articles in the Amiga Mail Tech Notes binder, just use the chapter and page numbers listed in the footer of each article page. The Tech Notes chapter is shown as a Roman numeral followed by the page number within that chapter.



For instance, the article at right would be filed in chapter III of the Amiga Tech Notes binder, "Exec". The article starts on page 99 of that chapter in the binder.

For more information on the Amiga Mail Tech Notes binder and the Commodore Developer Support Program, contact your local support manager or if you are in the U.S., contact CATS at (215) 431-9180.

Warning

The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

Disclaimer

This information is provided "As Is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any defect in the information, even if advised of the possibility of such damages.

This publication is printed on recycled material.

Amiga Mail News

New CATS Faces

Christian Ludwig has joined the CATS team as the Multimedia Standards Engineer. Christian has been actively using Amiga computers since their introduction and brings with him extensive experience in the design of user interfaces, interactive video programs, and multimedia software and hardware. Christian will help to keep existing multimedia standards up-to-date and will also aid in researching and developing new standards.

Another recent addition to the CATS staff is David Miller, who is the CATS Unix Support Engineer. He is responsible for supporting third party developers with technical assistance and product information pertaining to Commodore's Amiga Unix systems. David brings with him extensive experience with the Unix operating system and its standards, such as System V Release 4, POSIX, and the X/Open Common Applications Environment.

3rd Edition of the AmigaDOS Manual Slated for July Release

Bantam Books is finishing up the latest edition of the *AmigaDOS Manual*. The manual, now in its third edition, has been revised to cover release 2.0 AmigaDOS as well as release 1.3 AmigaDOS.

The expanded manual covers topics new to the *AmigaDOS Manual*, such as the release 2.0 Shell commands, the Fast Filing System, the standard DOS packets, and the DOS binary file hunks. The book also contains function descriptions for the over 140 functions present in the release 2.0 and release 1.3 versions of *dos.library*.

The book's improvements are not limited only to expanded AmigaDOS coverage. The new book has been reorganized to make locating information much quicker and easier. Unlike the old manual, which was actually three separate books in a single volume, the new manual is one single book (with one index) split into three sections: User's Manual section, The Developer's Manual section, and the Technical Reference section.

The manual is slated for release to bookstores in July.

IN THIS ISSUE

The ASL Library
John Orr

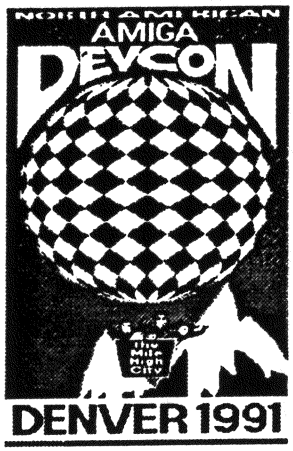
Standard Command Line Parsing
John Orr

Writing a Boopsi Image Class
David N. Junod

Boopsi's Transparent Base Classes
Peter Cherna

And We Have a Winner!

Congratulations to Scott Bennett of Expert Services. He designed the winning logo for the upcoming North American Developer's Conference. Scott's design was chosen by the CATS staff from a wide and varied field of entries. For his efforts, Scott will receive free admission to the conference, which will be held at the Denver Marriot City Center, in Denver, Colorado, from September 3-7.



If you are interested in attending the DevCon, registration forms are now available through CATS. CATS has arranged for substantial discounts on airfare and car rentals because *we want you to be there*. The registration form contains hotel and schedule information as well as a questionnaire on what sessions you would like to see in Denver.

If you haven't received a registration form or if you have any questions, please contact CATS administration at (215) 431-9180.

Australia to Host World of Commodore/Amiga

The first World of Commodore/Amiga in Australia is expected to attract many thousands of visitors to Darling Harbour in Sydney from July 12-14. The theme of the exhibition, which is being produced by the Hunter group, will be "A Focus on Creativity".

In addition to the exhibits, Commodore plans to sponsor seminars for resellers, developers, and end users. Also, a regional meeting for dealers from Australia, New Zealand, and Singapore will be held at the show.

Pat Byrne, Managing director, Commodore Australia, said that the show was planned because of an "overwhelming demand" from local end users. He said that the CDTV player will have its first viewing in Australia at the exhibition.

Show Manager Karen Jewell of the Hunter Group said that staging The World of Commodore/Amiga exhibition and seminar demonstrates Commodore's commitment to the local and regional computer market. "Commodore enjoys a 20 percent share of the Australian market with an installed base in excess of a quarter of a million machines," she said. "The region represents an enormous growth potential for third-party developers, and the enthusiasm of the existing user base indicates a rapidly growing market."

The ASL Library

by John Orr

The *asl.library* library provides the release 2.0 Amiga OS with system standard file and font requesters. Any application that needs to use a file or font requester should use *asl.library* requesters because they are easy to implement, they are easy for the user to learn, and because they are the same from one application to the next, they make applications more user friendly.

This article requires a general understanding of several Amiga concepts: TagItems and TagLists (see the Utility library Autodocs), Intuition screens and windows, some of the *graphics.library*'s font structures, and AmigaDOS pattern matching.

Opening a Simple File Requester

Using an ASL requester requires the use of three functions:

```
    myrequester = APTR AllocAslRequest( ULONG type, struct TagItem *tagList );
                  void FreeAslRequest( APTR request );
made_selection = BOOL AslRequest( APTR request, struct TagItem *tagList );
```

An ASL requester is based on a FileRequester or a FontRequester structure. Programs must allocate this structure with AllocAslRequest() (or AllocFileRequest()—see the *asl.library* Autodoc for more on this function). Currently, the type field (from the above prototype) can be one of two values (as defined in *<libraries/asl.h>*), `ASL_FileRequest`, to ask for a FileRequester structure, or `ASL_FontRequest`, to ask for a FontRequester structure. The values in this structure are for read access only. Any changes to them are carried out through *asl.library* function calls. The FreeAslRequest() function deallocates an ASL requester structure.

```
struct FileRequester    {   /* (from <libraries/asl.h>) */
        APTR    rf_Reserved1;
        BYTE    *rf_File;       /* Filename pointer             */
        BYTE    *rf_Dir;        /* Directory name pointer       */
        CPTR    rf_Reserved2;
        UBYTE   rf_Reserved3;
        UBYTE   rf_Reserved4;
        APTR    rf_Reserved5;
        WORD    rf_LeftEdge,rf_TopEdge; /* Preferred window pos */
        WORD    rf_Width,rf_Height;   /* Preferred window size  */
        WORD    rf_Reserved6;
        LONG    rf_NumArgs;     /* A-la WB Args, for multiselects */
        struct WBArg *rf_ArgList;
        APTR    rf_UserData;    /* Applihandle (you may write!!) */
        APTR    rf_Reserved7;
        APTR    rf_Reserved8;
        BYTE    *rf_Pat;        /* Pattern match pointer */
        };                      /* note - more reserved fields follow */
```

Based on the requester structure passed to it, AslRequest() brings up a font or file requester. When the user makes a selection, AslRequest returns a value. If it returns anything but FALSE (0), the user selected a file (or a font). AslRequest() returns FALSE if the user cancels the requester or the requester failed for some reason.

Both AslRequest() and AllocAslRequest() accept a tag list. This tag list is used to set the values in the FileRequester (or FontRequester) structure. The C code example *SimpleFR.c* at the end of this article uses the following tags (all the ASL tags are defined in *<libraries/asl.h>*):

```
ASL_Hail
ASL_Width
ASL_Height
ASL_LeftEdge
ASL_TopEdge
ASL_OKText
ASL_CancelText
ASL_File
ASL_Dir
```

ASL_Hail supplies a requester with a string to placed in the title bar of the requester window. ASL_Width, _Height, _LeftEdge, and _TopEdge describe the initial dimensions and position of the requester window. ASL_OKText and ASL_CancelText provide alternate strings for the "OK" and "Cancel" gadgets. Currently, the size of these gadgets limits the length of the names to about six characters. The two remaining tags from the above list, ASL_File and ASL_Dir, tags are specific to file requesters. They supply the initial file and directory names for the file requester.

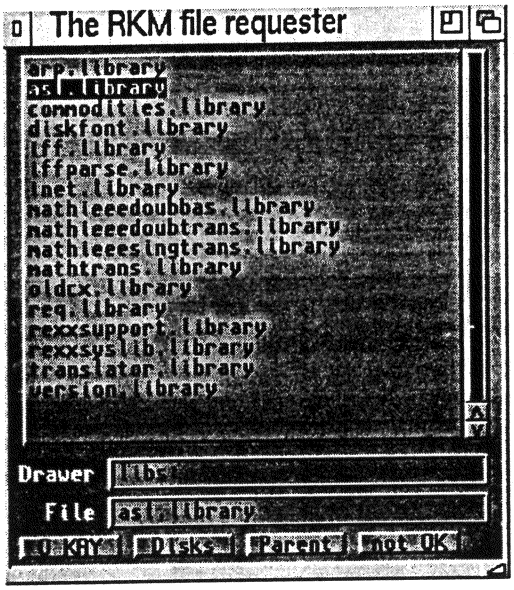


Figure 1 - The SimpleFR.c file requester

For a file requester, if AslRequest() returns anything but FALSE, *SimpleFR.c* looks at the FileRequester's rf_File and rf_Dir fields to get the name and directory of the file the user selected. Note that the requester allows the user to type in a name for the file and directory, which makes it possible for a file requester to return a file and directory that do not (currently) exist. In the case of a save requester (discussed a little later), the requester can create that non-existent directory.

The *simpleFR.c* example (and the rest of the ASL examples for that matter) may be a little misleading because it only uses the requester once. Any application that needs a requester should allocate the requester only once and pop up that requester *every* time the application needs one. If instead, the application allocated, used, and *deallocated* a requester each time it needed one, all of the settings the

user left in the requester (like the names the user last typed into the *drawer* or *file* string gadgets) would be reset to their initial values. By reusing a requester, the values stored in it are preserved.

Although retaining previous requester values from one request to another preserves user preferences, it also preserves other values stored in the requester structure. The file and font requester structures contain many private fields that toggle various features of the ASL requesters (those features are set via tags). If a program sets up a file requester with a specific feature, after the requester returns, the file requester structure is still set up with the specific feature. The next time the application wants to use that file requester *without* that specific feature, it must explicitly turn that feature off in the subsequent AslRequest() call.

File Pattern Matching and Multiple Selects

A file requester can filter out certain file and directory entries using *dos.library*'s "wildcarding". Turning on a file requester's "wildcarding" requires using the `ASL_FuncFlags` tag. This tag passes a bit field to a requester to set several options. The options are specific to the type of requester (file or font). The bitmasks are defined in *<libraries/asl.h>*.

The `ASL_FuncFlags`'s `FILF_PATGAD` flag toggles the file requester's "Pattern" gadget. The user can type an AmigaDOS wildcard pattern into this gadget that the requester uses to limit the file names that appear in the requester. An application can supply the "Pattern" gadget with a default pattern matching string using the `ASL_Pattern` tag.

Another feature of the ASL file requester is multiple selection. When an application sets up a file requester, it can set the `FILF_MULTISELECT` flag in the `ASL_FuncFlags` tag that allows the user to select several files at once.

When the user selects several files in the requester, the FileRequester's rf_NumArgs field contains the number of files selected. When the `FILF_MULTISELECT` bit is not set, rf_NumArgs is 0. If rf_NumArgs is not zero, rf_ArgList is a base pointer to an array of WBArg structures (defined in *<workbench/startup.h>*). There is a WBArg structure for each file the user selected.

The example *FilePat.c* illustrates multiple selection.

FilePat.c also introduces a new function and a new tag.

```
APTR AllocAslRequestTags( unsigned long type, Tag Tag1, ... );
BOOL AslRequestTags( APTR request, Tag Tag1, ... );
```

FilePat.c uses AllocAslRequestTags() instead of AllocAslRequest() to allocate and set up the file requester. AllocAslRequestTags() is an *Amiga.lib* function that accepts tag pairs in its parameter list, rather than a pointer to a tag list. AslRequestTags() also accepts tag pairs.

The ASL_Window Tag

An application that uses a custom screen normally wants its requesters to open on its screen. Using the ASL_Window tag, a program can associate a requester with a window so the requester appears on the window's screen. The ASL_Window tag is followed by a pointer to a window structure. ASL_Window works with both file and font requesters.

Normally, a requester associated with a window (using ASL_Window) uses the window's IDCMP port for its communication. An application may not want to share an IDCMP port with the requester. Using the ASL_FuncFlags tag, a program can ask for a requester that creates its own IDCMP port. There are two flags that accomplish this. The first, FILF_NEWIDCMP, is used on file requesters. The other, FONF_NEWIDCMP, is used on font requesters.

The Save Requester

The Save requester is a special type of file requester. The release 2.0 save requester differs from the regular file requester in several ways. The first is its appearance. The color of the text making up the file name list and the color of the backround the text is rendered on are swapped for the save requester. This makes the difference between the save and file requesters visual as well as functional. Another difference, is that a save requester does not allow the user to select by double-clicking on a file name. This prevents the user from accidentally double-clicking, and thus over-writing, the wrong file. The other difference is that if the user types a directory name into the save requester and the directory doesn't exist, the save requester will create that directory (after getting the user's permission via another requester).

The FILF_SAVE flag from the ASL_FuncFlags tag creates a save requester. Note that it does not make sense to have multiple selection in a save requester, so the FILF_SAVE flag overrides the FILF_MULTISELECT flag.

A Directory Requester

A program may not need to process files in a requester, only the directories. There is another variation on *asl.library*'s file requester that allows this. A new tag, ASL_ExtFlags1, toggles this option. ASL_ExtFlags1 passes a bit field like ASL_FuncFlags. One of ASL_ExtFlags1's flags, FIL1F_NOFILES, sets up a requester that has no string gadget for files and displays only directory names in the requester's scrolling list. When AslRequest() (or AslRequestTags()) returns successfully, the FileRequester structure's rf__Dir field contains the name of the directory the user selected.

Currently, there is another flag defined for `ASL_ExtFlags1`, `FIL1F_MATCHDIRS`. If file pattern matching is on (see the `FILF_PATGAD` flag for `ASL_FuncFlags`), setting `FIL1F_MATCHDIRS` tells the file requester to pattern match directory names as well as file names. Of course, if both of `ASL_ExtFlags1`'s flags are set, the requester will only pattern match directory names.

The Font Requester

The ASL library also contains a font requester. The default font requester consists of two scrolling lists, one containing font names, the other containing font sizes. Each scrolling list is accompanied by a string gadget which lets the user type in the name and size of the font they want.

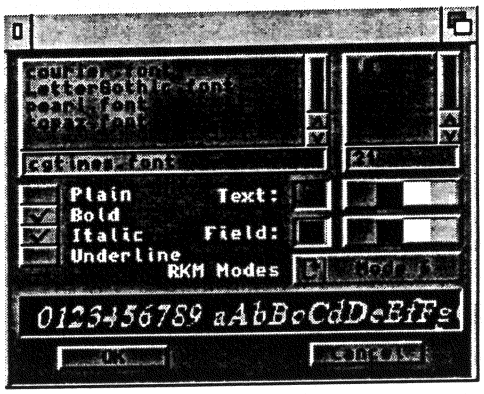


Figure 2 - The FontReq.c font requester

The font size string gadget (actually, an integer gadget) is especially useful because it allows the user to type in a font size that doesn't appear in the font size scrolling list. If the font size the user requests does not already exist in the system, the Amiga can generate the font size the user requests either through bitmap scaling, or, if scalable outline fonts are available, by scaling an outline font.

Using the font requester is similar to using the file requester. First, allocate a requester structure. A font requester requires the FontRequester structure:

```
struct FontRequester    {
        APTR    fo_Reserved1[2];
        struct TextAttr fo_Attr;        /* Returned TextAttr            */
        UBYTE   fo_FrontPen;            /* Returned pens, if selected   */
        UBYTE   fo_BackPen;
        UBYTE   fo_DrawMode;
        APTR    fo_UserData;
        };
```

To allocate this structure, use AllocAslRequest() or AllocAslRequestTags() with the first argument set to `ASL_FontRequest`.

There are several tags that are specific to the font requester:

```
ASL_FontName
ASL_FontHeight
ASL_FontStyles
ASL_FontFlags
ASL_FrontPen
ASL_BackPen
ASL_MinHeight
ASL_MaxHeight
ASL_ModeList
```

The first six tags are used to place initial values into the FontRequester structure. They correspond respectively to the FontRequester's fo_Attr.ta_Name, fo_Attr.ta_YSize, fo_Attr.ta_Style, fo_Attr.ta_Flags, fo_FrontPen, and fo_BackPen fields. `ASL_MinHeight` specifies the minimum Y height that the font requester should list. `ASL_MaxHeight` specifies the maximum Y height that the font requester should list. Note that these tags only limit the range of font sizes that the font requester displays, the user is free to type in *any* value. The `ASL_ModeList` tag specifies the strings used for the font requester's drawing mode gadget. This is discussed later in the chapter.

The are several options available to the font requester that are toggled by setting flags with the `ASL_FuncFlags` tag. These flags are:

```
FONF_FRONTCOLOR
FONF_BACKCOLOR
FONF_STYLES
FONF_FIXEDWIDTH
FONF_DRAWMODE
```

A simple font requester (one without any of the above `FONF_` flags set) only lets the user choose a font and a Y size. Setting the flags above adds options to the font requester. `FONF_FRONTCOLOR` and `FONF_BACKCOLOR` add color selection gadgets to the requester, one for choosing a font's foreground color (labeled "Text" on the requester) and the other for choosing the backround color (labeled "Field"). The font requester records the user's setting in the FontRequester's fo_FrontPen and fo_BackPen fields. `FONF_STYLES` sets up several gadgets to choose the style of the font (bold, italics, underline). The font requester saves these setting in the fo_Attr.ta_Style bit field according to the style flags defined in *<graphics/text.h>*. `FONF_FIXEDWIDTH` limits the font name display to fixed width (non-proportional) fonts (note that this does not prevent the user from typing in a proportional font name). `FONF_DRAWMODE` adds a cycle gadget to the font requester so the user can choose the draw mode. The draw mode is saved in the requester's fo_DrawMode field. The number stored there corresponds to the draw mode's position in the gadget's cycle.

The draw mode cycle gadget initially is labeled "Mode" and has three elements in its cycle: "JAM1", "JAM2", and "Complement". These yield a result of 0, 1, and 2, respectively. It is possible to change the names and number of draw modes with the `ASL_ModeList` tag. This tag accepts a pointer to an array of strings. The first string replaces "Mode" as the label for the draw mode cycle gadget. The strings that follow replace the elements of the cycle gadget. The last entry in the array has to be `NULL` to tell the requester where the list of entries ends.

The *FontReq.c* example illustrates how to use a font requester.

The ASL_HookFunc Tag

The ASL_HookFunc tag passes an ASL requester a pointer to a custom function. The requester can use this function for two purposes. The first is to determine if the requester should display a particular file or font name. The other purpose is to process messages that the requester receives at its IDCMP port that are not meant for the requester. Some ASL_FuncFlags flags toggle these options:

```
FILF_DOWILDFUNC
FONF_DOWILDFUNC
FILF_DOMSGFUNC
FONF_DOMSGFUNC
```

The FILF_DOWILDFUNC and FONF_DOWILDFUNC flags cause a requester to call the hook function for every file or font entry. The requester displays the file or font name only if the hook function tells it to. For a file requester, if the hook function returns a *zero*, the file requester will display the file name. For a font requester, if the hook function returns *anything but zero*, the font requester will display the font name and size. Note that if the DOWILDFUNC function changes the current directory, it must restort the current directory before it exits. Neglecting to restore the current directory can confuse DOS.

The FILF_DOMSGFUNC and FONF_DOMSGFUNC flags cause a requester to call the hook function when it receives certain IntuiMessages at the IDCMP port it shares with a window (see the ASL_Window tag). The requester passes on IntuiMessages not meant for the requester. The hook function is responsible for returning a pointer to the IntuiMessage. The requester will take care of replying to the message.

A requester passes three parameters to the hook function:

```
ULONG MyHookFunc(ULONG type, CPTR object, CPTR AslRequester)
```

If MyHookFunc() is called from a file requester doing DOWILDFUNC:

type = FILF_DOWILDFUNC
object is a pointer to an AnchorPath structure (from <*dos/dosasl.h*>)
AslRequester is a pointer to the FileRequester that called the hook function
(Return a zero to display this file)

Note that the AnchorPath structure is a *dos.library* structure used in pattern matching. See *dos.library* documentation for more details.

Referencing the specific file or directory from within the DOWILDFUNC hook function is a little tricky. For a file requester, the rf_Dir field is not guaranteed to contain the name of the directory being scanned. The correct way to find the file and directory name is from the AnchorPath structure passed to the DOWILDFUNC. The structure contains a FileInfoBlock structure called ap_Info that contains the file name (fib_FileName). The AnchorPath structure's ap_Current field contains a pointer to an AChain structure (from <*dos/dosasl.h*>) which contains a field called an_Lock. This is a lock on the directory being scanned.

If MyHookFunc() is called from a font requester doing DOWILDFUNC:

type = FONF_DOWILDFUNC
object is a pointer to a TextAttr structure (from *<graphics/text.h>*)
AslRequester is a pointer to the FontRequester that called the hook function
(Return non-zero to display this particular font size)

If MyHookFunc() is called from a *file or font* requester doing DOMSGFUNC:

type = FILF_DOMSGFUNC
object is a pointer to the IntuiMessage for the function to process
AslRequester is a pointer to the *FileRequester or FontRequester* that called the hook function
(Return the pointer to the IntuiMessage)

Notice that it is possible for a requester to use both DOWILDFUNC and DOMSGFUNC at the same time. The hook function has to differentiate between the two cases by testing the type passed to it. It is not possible for font and file requester to share a hook function for a DOWILDFUNC, because FILF_DOWILDFUNC is defined to be the same value as FONF_DOWILDFUNC, so the hook function cannot tell if object (from the prototype above) is a pointer to an AnchorPath structure or a pointer to a TextAttr structure. It *is* possible for font and file requesters to share one hook function for DOMSGFUNC (even though FILF_DOMSGFUNC and FONF_DOMSGFUNC are equal) because, in this case, font and file requesters both call the hook function in the same manner.

The example *FileHook.c* illustrates the use of a hook function for both DOWILDFUNC and DOMSGFUNC.

```
;/* simpleFR.c - Execute me to compile me with SASC 5.10
LC -b1 -cfistq -v -y -j73 simpleFR.c
Blink FROM LIB:c.o,simpleFR.o TO simpleFR LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/

#include <clib/asl_protos.h>
#include <clib/exec_protos.h>
#include <clib/alib_stdio_protos.h>
#include <dos/dosasl.h>
#include <exec/libraries.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif

UBYTE *vers = "\0$VER: simpleFR 1.1";

void main(void);

#define MYLEFTEDGE 0
#define MYTOPEDGE  0
#define MYWIDTH    320
#define MYHEIGHT   400

struct Library *AslBase;

struct TagItem frtags[] =
{
    ASL_Hail,       (ULONG)"The RKM file requester",
    ASL_Height,     MYHEIGHT,
    ASL_Width,      MYWIDTH,
    ASL_LeftEdge,   MYLEFTEDGE,
    ASL_TopEdge,    MYTOPEDGE,
    ASL_OKText,     (ULONG)"O KAY",
    ASL_CancelText, (ULONG)"not OK",
    ASL_File,       (ULONG)"asl.library",
    ASL_Dir,        (ULONG)"libs:",
    TAG_DONE
};


void main()
{
    struct FileRequester *fr;

    if (AslBase = OpenLibrary("asl.library", 36L))
    {
        if (fr = (struct FileRequester *)
            AllocAslRequest(ASL_FileRequest, frtags))
        {
            /* Application body, blah, blah,... */

            /* Application need a requester */
            if (AslRequest(fr, 0L))
                printf("file choice = %s%s\n", fr->rf_Dir, fr->rf_File);
            else
                printf("User Cancelled\n");

            /* More application body, blah, blah... */
        }

        /* Don't need any more requesters, better
        ** give the requester structure back.
        */
        FreeAslRequest(fr);
    }
    CloseLibrary(AslBase);
}
```

```
;/* filepat.c - Execute me to compile me with SASC 5.10
LC -b1 -cfistq -v -y -j73 filepat.c
Blink FROM LIB:c.o,filepat.o TO filepat LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/

#include <clib/asl_protos.h>
#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/alib_stdio_protos.h>
#include <workbench/startup.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <graphics/displayinfo.h>
#include <exec/libraries.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif

UBYTE *vers = "\0$VER: filepat 1.0";

void main(void);

struct Library *AslBase;
struct IntuitionBase *IntuitionBase;
struct Screen *screen;
struct Window *window;

struct WBArg *wbargs;

LONG x;


void main()
{
    struct FileRequester *fr;

    if (AslBase = OpenLibrary("asl.library", 36L))
    {
        if (IntuitionBase = (struct IntuitionBase *)
                OpenLibrary("intuition.library", 36L))
        {
            if (screen = (struct Screen *)OpenScreenTags(NULL,
                    SA_DisplayID, HIRESLACE_KEY,
                    SA_Title, "ASL Test Screen",
                    TAG_END))
            {
                if (window = (struct Window *)OpenWindowTags(NULL,
                        WA_CustomScreen, screen,
                        WA_Title, "ASL Test Window",
                        WA_Flags, WINDOWDEPTH | WINDOWDRAG,
                        TAG_END))
                {
                    if (fr = (struct FileRequester *)
                        AllocAslRequestTags(ASL_FileRequest,
                            ASL_Hail, (ULONG)"RKM File Requester, FilePat",
                            ASL_Dir,  (ULONG)"libs:",
                            ASL_File, (ULONG)"asl.library",

                            /* The initial pattern string for the
                            ** pattern gadget.
                            */
                            ASL_Pattern, (ULONG)"~(rexx#?|math#?)",

                            /* turn on multiselection and the pattern
                            ** matching gadget.
                            */
                            ASL_FuncFlags, FILF_MULTISELECT | FILF_PATGAD,

                            /* This requester is associated with this
                            ** window (and uses its message port).
                            */
                            ASL_Window, window,
                            TAG_DONE))
```

```
                {
                    /* Application code body...*/

                    /* Put up file requester */
                    if (AslRequest(fr, 0L))
                    {
                        /* If the file requester's rf_NumArgs field
                        ** is not zero, the user multiselected. The
                        ** number of files is stored in rf_NumArgs.
                        */
                        if (fr->rf_NumArgs)
                        {
                            /* rf_ArgList is an array of WBArg structures
                            ** (defined in <workbench/startup.h>).
                            ** Each entry in the WBArg array corresponds
                            ** to one of the files the user selected
                            ** (the entries are in alphabetical order).
                            */
                            wbargs = fr->rf_ArgList;

                            /* The user multiselected, step through
                            ** the list of selected files.
                            */
                            for ( x=0;  x < fr->rf_NumArgs;  x++ )
                                printf("Argument %d - %s%s\n", x,
                                    fr->rf_Dir, wbargs[x].wa_Name);
                        }
                        else
                            /* The user didn't multiselect, use the
                            ** normal way to get the file name.
                            */
                            printf("%s%s\n", fr->rf_Dir, fr->rf_File);
                    }
                    /* More application code body... */

                    /* Done with the FileRequester, better return it */
                    FreeAslRequest(fr);
                }
                CloseWindow(window);
            }
            CloseScreen(screen);
        }
        CloseLibrary(IntuitionBase);
    }
    CloseLibrary(AslBase);
  }
}
```

```
;/* fontreq.c - Execute me to compile me with Lattice 5.10
LC -bl -cfistq -v -y -j73 fontreq.c
Blink FROM LIB:c.o,fontreq.o TO fontreq LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/

#include <clib/asl_protos.h>
#include <clib/exec_protos.h>
#include <clib/alib_stdio_protos.h>
#include <exec/libraries.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif


UBYTE *vers = "\0$VER: fontreq 1.0";

void main(void);
struct Library *AslBase, *UtilityBase;

/* The replacement strings for the "mode" cycle gadget.  The
** first string is the cycle gadget's label.  The other strings
** are the actual strings that will appear on the cycle gadget.
*/
UBYTE *modelist[] =
{
    "RKM Modes",
    "Mode 0",
    "Mode 1",
    "Mode 2",
    "Mode 3",
    "Mode 4",
    "Mode 5",
    "Mode 6",
    "Mode 7",
    "Mode 8",
    "Mode 9",
    NULL
};


void main()
{

    struct FontRequester *fr;

    if (AslBase = OpenLibrary("asl.library", 36L))
    {

        if (fr = (struct FontRequester *)
            AllocAslRequestTags(ASL_FontRequest,
                /* tell the requester to use my custom mode names */
                ASL_ModeList, modelist,

                /* Supply initial values for requester */
                ASL_FontName, (ULONG)"topaz.font",
                ASL_FontHeight, 11L,
                ASL_FontStyles, FSF_BOLD | FSF_ITALIC,
                ASL_FrontPen,  0x00L,
                ASL_BackPen,   0x01L,

                /* Only display font sizes between 8 and
                ** 14, inclusive. */
                ASL_MinHeight, 8L,
                ASL_MaxHeight, 14L,

                /* Give us all the gadgetry, but only display
                ** fixed width fonts */
                ASL_FuncFlags, FONF_FRONTCOLOR | FONF_BACKCOLOR |
                    FONF_DRAWMODE | FONF_STYLES | FONF_FIXEDWIDTH,
                TAG_DONE))
        {
            /* application code here... */
```

```
            /* Pop up the requester */
            if (AslRequest(fr, 0L))
            {
                /* The user selected something,  report their choice */
                printf("%s\n  YSize = %d  Style = 0x%x   Flags = 0x%x\n"
                       "  FPen = 0x%x   BPen = 0x%x   DrawMode = 0x%x\n",
                            fr->fo_Attr.ta_Name,
                            fr->fo_Attr.ta_YSize,
                            fr->fo_Attr.ta_Style,
                            fr->fo_Attr.ta_Flags,
                            fr->fo_FrontPen,
                            fr->fo_BackPen,
                            fr->fo_DrawMode);
            }
            else
                /* The user cancelled the requester, or
                ** some kind of error occured preventing
                ** the requester from opening. */
                printf("Request Cancelled\n");

            /* more application code here ...*/

            FreeAslRequest(fr);
        }
        CloseLibrary(AslBase);
    }
}
```



```
:/* filehook.c - Execute me to compile me with Lattice 5.10
LC -bl -cfistq -v -y -j73 filehook.c
Blink FROM LIB:c.o,filehook.o TO filehook LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/

#include <clib/asl_protos.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/intuition_protos.h>
#include <clib/alib_stdio_protos.h>
#include <dos/dosasl.h>
#include <intuition/intuition.h>
#include <exec/libraries.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }  /* really */
#endif

#define DESTPATLENGTH 20

UBYTE *vers = "\0$VER: filehook 1.0";

void main(void);

struct Library *AslBase;
struct IntuitionBase *IntuitionBase;
struct Window *window;

CPTR HookFunc();

/* this is the pattern matching string that the hook function uses */
UBYTE *sourcepattern = "(#?.info)";
UBYTE pat[DESTPATLENGTH];


void main()
{

    struct FileRequester *fr;

    /* This is a dos.library function that turns a pattern matching
    ** string into something the DOS pattern matching functions
    ** can understand.
    */
    ParsePattern(sourcepattern, pat, DESTPATLENGTH);

    if (AslBase = OpenLibrary("asl.library", 36L))
    {
        if (IntuitionBase = (struct IntuitionBase *)
                    OpenLibrary("intuition.library", 36L))
        {
            /* open a window that gets ACTIVEWINDOW events */
            if (window = (struct Window *)OpenWindowTags(NULL,
                    WA_Title, "ASL Hook Function Example",
                    WA_IDCMP, IDCMP_ACTIVEWINDOW,
                    WA_Flags, WINDOWDEPTH,
                    TAG_END))
            {
                if (fr = AllocFileRequest())
                {
                    /* application body here... */


                    if (AslRequestTags(fr,
                        ASL_Window, window,
                        ASL_TopEdge, 0L,
                        ASL_Height, 200L,
                        ASL_Hail, (ULONG)"Pick an icon to save",
                        ASL_HookFunc, (ULONG)HookFunc,
                        ASL_FuncFlags, FILF_DOWILDFUNC | FILF_DOMSGFUNC | FILF_SAVE,
                        ASL_OKText, (ULONG)"Save",
                        TAG_DONE))
                    {
                        printf("You picked %s%s\n", fr->rf_Dir, fr->rf_File);
                    }
```

```
                    /* more application body here */

                    FreeFileRequest(fr);
                }
                CloseWindow(window);
            }
            CloseLibrary(IntuitionBase);
        }
        CloseLibrary(AslBase);
    }
}


CPTR HookFunc(LONG type, CPTR obj, struct FileRequester *fr)
{
    static BOOL returnvalue;
    switch(type)
    {
        case FILF_DOMSGFUNC:
        /* We got a message meant for the window */
            printf("You activated the window\n");
            return(obj);
            break;
        case FILF_DOWILDFUNC:
        /* We got an AnchorPath structure, should
        ** the requester display this file? */

            /* MatchPattern() is a dos.library function that
            ** compares a matching pattern (parsed by the
            ** ParsePattern() DOS function) to a string and
            ** returns true if they match. */
            returnvalue = MatchPattern(pat,
                    ((struct AnchorPath *)obj)->ap_Info.fib_FileName);

            /* we have to negate MatchPattern()'s return value
            ** because the file requester expects a zero for
            ** a match not a TRUE value */
            return( (CPTR)(! returnvalue) );
            break;
    }
}
```

Standard Command Line Parsing

by John Orr

One of the new features in release 2.0 is system standard command line parsing. Its presence has two benefits: it standardizes the way in which the user supplies command line arguments, making it much easier on the user, and it also removes some tedious programming (and code size) from every application that uses it.

The *dos.library*'s ReadArgs() routine is the heart of this feature:

```
struct RDArgs *rda = ReadArgs( UBYTE *argtemplate, LONG *argarray, struct RDArgs *myrda );
```

This function stores each argument supplied on the command line in its corresponding entry in the array of LONG words, argarray. The format in which ReadArgs() stores each argument is based on the description of the command line supplied in the first argument, argtemplate. This description is a C-style string containing a name for each argument.

Each argument name in the template should be a full, descriptive name (for example "Quick" not "Q"). Each option can be prepended by an abbreviation of the form "abbrev=" (for example "Q=Quick"). The argtemplate options must be delimited by commas. Avoid using the names of common commands as keywords, otherwise the user will have to delimit them with quotes.

The ordinal position that an argument appears in the description string determines what its corresponding position in argarray is (the first argument corresponds to the first entry in argarray, the second argument corresponds to the second entry in argarray, ...). There must be an entry in argarray for each argument in the description string so that ReadArgs() has a place to store each argument's value.

Each argument name in the template can be followed by modifiers that tell ReadArgs() the format of the argument. The valid modifiers are:

> /S - Switch. This is considered a boolean variable. If this option is present, ReadArgs() will set the corresponding array entry in argarray (an array of LONGs) to something besides zero. If the option is not present, ReadArgs() will set the entry to 0.
>
> /K - Keyword. This means that ReadArgs() will not fill in the corresponding entry in argarray unless the keyword appears with the parameter. For example, if the template is "Name/K", then unless "Name=<string>" or "Name <string>" appears in the command line, the "Name" entry in argarray will not be altered by ReadArgs().

/N - Number. This means the parameter is considered a decimal integer, and ReadArgs() will convert it to a LONG. If the argument is not valid, ReadArgs() will fail. If the option is present in the command line (and it is valid), ReadArgs() will fill in the corresponding entry with a pointer to the LONG.

/T - Toggle. This is similar to the switch (/S) modifier, but causes the corresponding boolean (in argarray) to toggle. For example, if the array entry corresponding to an argtemplate of "binary\T" is set to something besides FALSE and the word "binary" appears by itself on the command line, ReadArgs() will toggle that array entry to FALSE.

/A - Always. This modifier tells ReadArgs() that this option is required. ReadArgs() will fail if the keyword does not appear in the command line.

/F - Final (rest of line). If this is specified, the part of the command line that follows this option is taken as the parameter for this option, even if other option keywords appear in it.

/M - Multiple arguments. This means the argument will take any number of strings (or integers as this modifier can be used with the /N modifier), returning them as an array of strings. Any arguments not considered to be part of another option will be added to this option. Only one /M should appear in a template. Example: for a template "Dir/M,All/S" the command-line "foo bar all qwe" will set the boolean "all", and return an array consisting of "foo", "bar", and "qwe". The entry in the array will be a pointer to an array of string pointers, the last of which will be NULL.

There is an intentional interaction between /M parameters and /A parameters. If there are unfilled /A parameters after parsing, ReadArgs() will grab strings from the end of a previous /M parameter list to fill the /A's. This is used for things like Copy ("From/A/M,To/A").

If the user does not supply a non-required argument (one without the "/A" modifier) on the command line, ReadArgs() will leave the argument's argarray entry alone. Before calling ReadArgs(), a program should either set the argarray entries to reasonable default values or clear them, so the application can't be confused by any garbage values left in the array.

If it is successful, ReadArgs() returns a pointer to a RDArgs structure (from *<dos/rdargs.h>*). ReadArgs() uses this structure internally to control its operation. It is possible to pass ReadArgs() a custom RDArgs structure (myrda in the ReadArgs() prototype above). For most applications myrda will be NULL, as most applications do not need to control ReadArgs().

```
struct RDArgs {
    struct    CSource RDA_Source; /* Select input source */
    LONG      RDA_DAList;         /* PRIVATE. */
    UBYTE     *RDA_Buffer;        /* Optional string parsing space. */
    LONG      RDA_BufSiz;         /* Size of RDA_Buffer (0..n) */
    UBYTE     *RDA_ExtHelp;       /* Optional extended help */
    LONG      RDA_Flags;          /* Flags for any required control */
};
```

Any successful call to ReadArgs() (even those that use a custom RDArgs structure) must be complemented with a call to FreeArgs() to free the resources that ReadArgs() allocates:

```
void FreeArgs(struct RDArgs *rda);
```

where rda is the RDArgs structure used by ReadArgs().

An application can use a custom RDArgs structure to provide an alternate command line source, an alternate temporary storage buffer, or an extended help string. The custom RDArgs structure must be allocated with AllocDosObject() and deallocated with FreeDosObject(). See the Autodocs for more details on these functions.

The RDArgs.RDA_Source field is used to supply ReadArgs() with an alternate command line to parse. If this field is non-NULL, ReadArgs() will use it as a pointer to a CSource structure describing the alternate command line. The CSource structure (from *<dos/rdargs.h>*) is as follows:

```
struct CSource {
    UBYTE   *CS_Buffer;
    LONG    CS_Length;
    LONG    CS_CurChr;
};
```

Where CS_Buffer is the command line to parse, CS_Length is the length of CS_Buffer, and CS_CurChr is the position in CS_Buffer from which ReadArgs() should begin its parsing. Normally CS_CurChr is initialized to zero.

ReadArgs() uses the RDArgs structure's RDA_DAList field for internal use. This field *must* be set to NULL before ReadArgs() uses this structure.

The RDA_Buffer and RDA_BufSiz fields allow an application to supply a fixed-size buffer in which to store parsed data. This allows the application to pre-allocate a buffer rather than requiring ReadArgs() to allocate buffer space. If either RDA_Buffer or RDA_BufSiz is NULL, ReadArgs() assumes the application has not supplied a buffer.

RDA_ExtHelp is a text string which ReadArgs() displays if the user asks for additional help. The user asks for additional help by typing a question mark when ReadArgs() prompts the user for input (which normally happens only after he or she types a question mark as the only argument on the command line).

RDA_Flags is a bit field used to toggle certain options of ReadArgs(). Currently, only one option is implemented, RDAF_NOPROMPT. When set, RDAF_NOPROMPT prevents ReadArgs() from prompting the user.

The following code, *ReadArgs.c*, uses a custom RDArgs structure to pass a command line to ReadArgs.

```
;/* ReadArgs.c - Execute me to compile me with Lattice 5.10a
LC -bl -cfistq -v -y -j73 ReadArgs.c
Blink FROM LIB:c.o,ReadArgs.o TO ReadArgs LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit
*/


#include <dos/dos.h>
#include <dos/rdargs.h>
#include <clib/dos_protos.h>
#include <clib/alib_stdio_protos.h>


#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disable Lattice CTRL/C handling */
int chkabort(void) { return(0); }
#endif
```

```
UBYTE *vers = "\0$VER: ReadArgs 1.0";

#define TEMPLATE "S=SourceFiles/A/M,D=DebugLevel/K/N,L=link/S"
#define OPT_SOURCE  0
#define OPT_DEBUG   1
#define OPT_LINK    2
#define OPT_COUNT   3

/* The array of LONGs where ReadArgs() will store the data from
** the command line arguments.  C guarantees that all the array
** entries will be set to zero.
*/
LONG result[OPT_COUNT];

/* My custom RDArgs */
struct RDArgs *myrda;

ULONG StrLen(UBYTE *);

void main(void)
{
    UWORD x;
    UBYTE **sourcefiles;

    /* Need to ask DOS for a RDArgs structure */
    if (myrda = (struct RDArgs *)AllocDosObject(DOS_RDARGS, NULL))
    {
        /* set up my parameters for ReadArgs() */

        /* use the following command line */
        myrda->RDA_Source.CS_Buffer = "file1 file2 file3 D=1 Link file4 file5\n";
        myrda->RDA_Source.CS_Length = (LONG)StrLen(myrda->RDA_Source.CS_Buffer);

        /* parse my command line */
        if (ReadArgs(TEMPLATE, result, myrda))
        {
            /*start printing out the results */

            /* We don't need to check if there is a value in
            ** result[OPT_SOURCE] because the ReadArgs() template
            ** requires (using the /A modifier) that there be
            ** file names, so ReadArgs() will either fill in a
            ** value or ReadArgs() will fail.
            */
            sourcefiles = (UBYTE **)result[OPT_SOURCE];
            /* VPrintf() is a lot like Printf() except it's in
            ** ROM, and the arguments are referenced from an
            ** array rather than being extracted from the stack.
            */
            VPrintf("Files specified:\n", NULL);
            for (x=0; sourcefiles[x]; x++)
                VPrintf("\t%s\n", (LONG *)&sourcefiles[x]);

            /* Is there something in the "DebugLevel" option?
            ** If there is, print it.
            */
            if (result[OPT_DEBUG])
                VPrintf("Debugging Level = %ld\n", (LONG *)result[OPT_DEBUG]);

            /* If the link toggle was present, say something about it. */
            if (result[OPT_LINK])
                VPrintf("linking...\n", NULL);
            FreeArgs(myrda);
        }
        FreeDosObject(DOS_RDARGS, myrda);
    }
}

ULONG StrLen(UBYTE *string)
{
    ULONG x = 0L;

    while (string[x++]);
    return(x);
}
```

Writing a Boopsi Image Class

By David N. Junod

Editor's note: this article and its source code reference several functions that, at press time, were only available from the classface.asm and hookface.asm assembly source files that appear on the Atlanta and Milan DevCon disks. The functions from these files should eventually appear in amiga.lib.

The most sophisticated level of Intuition programming is to write a boopsi class. The easiest way to enter the boopsi class writer's arena is to write an image class.

Boopsi's *imageclass* is one of the standard classes built into Intuition. As its name implies, it is a class of Intuition Images. These boopsi images can be used in place of traditional Image structure (as they contain an Intuition Image structure), but they are much more powerful. By using boopsi methods, an application or Intuition can tell an *imageclass* object to render itself. Because it renders itself (rather than Intuition rendering it), the *imageclass* object is free to render whatever it wants (well, within reason). For example, a boopsi image object can render itself according to the current display resolution, or to scale itself to any size an application requests.

This article is designed to provide the novice boopsi programmer with the information needed to write an image class for their application. This article assumes the reader is already familar with some boopsi concepts. For more information on boopsi, see the article "Introduction to Boopsi" in the March/April 1991 issue of Amiga Mail or the Atlanta (or Milan) DevCon notes and disks. The example custom class at the end of this article, *mytextlabelclass.c*, shows how to create a custom image class that renders a text label with an underline beneath a character. This character can be used to trigger some event.

When designing a specific class, you must first choose a superclass that is suitable for your needs. If you are creating a new image class, then its superclass will either be *imageclass* or some subclass of *imageclass*.

Classes may be public or private. Any application can access a public class. Before a class can be considered public it must first have a name and must be part of the public class list maintained by

Intuition. It can then be accessed by other applications. A private class is not in the public class list and can only be used by applications that have a pointer to the Class structure (usually the application that implemented the class).

Callback Hooks

When you present a custom image to Intuition, you provide a pointer to a Hook structure that Intuition uses to find functions needed by various image operations. Without going into great detail, a hook provides a pointer to a function that the system calls using Amiga register parameter conventions. The hook supplies enough information to conveniently transfer control to a High-Level Language (HLL) entry point. Boopsi *imageclass* objects provide Intuition with a hook to a method dispatcher function.

The Hook structure is defined as follows (from *<utility/hooks.h>*):

```
/* new standard hook structure */
struct Hook
{
    struct MinNode  h_MinNode;
    ULONG           (*h_Entry)();    /* stub function entry point */
    ULONG           (*h_SubEntry)();/* the custom function entry point */
    VOID            *h_Data;         /* owner specific */
}
```

The assembly language stub for C parameter conventions that boopsi classes (and custom gadgets) require is:

```
_hookEntry:
    move.l  a1,-(sp)                ; push message packet pointer
    move.l  a2,-(sp)                ; push object pointer
    move.l  a0,-(sp)                ; push hook pointer
    move.l  h_SubEntry(a0),a0       ; fetch C entry point ...
    jsr     (a0)                    ; ... and call it
    lea     12(sp),sp               ; fix stack
    rts
```

The C language stub, for C compilers that support registerized parameters is:

```
/* This function converts register-parameter hook calling
 * convention into standard C conventions.  It requires a C
 * compiler that supports registerized parameters, such as
 * SAS/C 5.xx or greater.
 */
ULONG __asm hookEntry(
    register __a0 struct Hook *h,
    register __a2 VOID *o,
    register __a1 VOID *msg)
{
    return ((*h->h_SubEntry)(h, o, msg));
}
```

Initializing a Boopsi Class

You need some simple code to initialize a class and its hook. When initializing a class, you specify the size of the class's instance and what the superclass is, and you also have to supply a pointer to a hook entry stub.

The following code fragment illustrates the initialization of a private subclass of *imageclass*.

```
ULONG __saveds dispatchmyTextLabel();

/* This is the data that each instance of our class will need. */
struct localObjData
{
    /* Font to use */
    struct TextFont *lod_Font;

    /* The key that is underlined */
    UWORD lod_Key;

    /* DrawMode */
    UBYTE lod_Mode;
}

#define MYCLASSID           NULL
#define SUPERCLASSID        (IMAGECLASS)
#define LSIZE               (sizeof(struct localObjData))

Class *initmyTextLabelClass (VOID)
{
    extern ULONG __saveds dispatchmyTextLabel();
    extern ULONG hookEntry ();        /* defined in hookface.asm */
    Class *cl;

    if (cl = MakeClass (MYCLASSID, SUPERCLASSID, NULL, LSIZE, 0))
    {
        /* Fill in the callback hook */
        cl->cl_Dispatcher.h_Entry = hookEntry;
        cl->cl_Dispatcher.h_SubEntry = dispatchmyTextLabel;
    }

    /* Return a pointer to the class */
    return (cl);
}
```

In order to make the class public instead of private, do the following:

```
#define MYCLASSID           "mytextlabelclass"

ULONG __saveds dispatchmyTextLabel();

Class *initmyTextLabelClass (VOID)
{
    extern ULONG __saveds dispatchmyTextLabel();
    extern ULONG hookEntry ();
    Class *cl;

    if (cl = MakeClass (MYCLASSID, SUPERCLASSID, NULL, LSIZE, 0))
    {
        /* Fill in the callback hook */
        cl->cl_Dispatcher.h_Entry = hookEntry;
        cl->cl_Dispatcher.h_SubEntry = dispatchmyTextLabel;

        /* Make the class public */
        AddClass (cl);
    }

    /* Return a pointer to the class */
    return (cl);
}
```

Boopsi Dispatcher

Now all you need to do is implement a dispatcher routine. When the dispatcher is in operation, Intuition passes method IDs to it. The dispatcher will either execute code corresponding to the a method ID (the code is usually part of the dispatcher) or delegate processing the method to the superclass (or it can do a little of both).

The following fragment provides an example of what a dispatcher for a boopsi class looks like (Note that __saveds (Save DS) is used to insure that register A4 is set up properly for indirect addressing with the SASC compiler):

```
ULONG __saveds dispatchmyTextLabel (Class *cl, Object *o, Msg msg)
{
    struct localObjData *lod;
    Object *newobj;
    ULONG retval;

    switch (msg->MethodID)
    {
        /* Create a new object */
        case OM_NEW:
            /* Have our superclass create it.  DSM() passes on the message
             * to the superclass, where msg is the structure containing the
             * message specific parameters.
             */
            if (newobj = (Object *) DSM (cl, o, msg))
            {
                /* Set the attributes */
                setmyTextLabelAttrs(cl, newobj, (struct opSet *) msg);
            }

            retval = (ULONG) newobj;
            break;

        /* Obtain information on an attribute */
        case OM_GET:
            retval = getmyTextLabelAttrs (cl, o, (struct opGet *) msg);
            break;

        /* Set attributes */
        case OM_UPDATE:
        case OM_SET:
            /* Let the superclass set the attributes that it
             * knows about. */
            retval = DSM (cl, o, msg);

            /* Set the attributes that we care about */
            retval |= setmyTextLabelAttrs (cl, o, (struct opSet *) msg);
            break;

        /* Draw the various states that the image supports */
        case IM_DRAW:
        case IM_DRAWFRAME:
            retval = drawmyTextLabel (cl, o, (struct impDraw *) msg);
            break;

        /* Let the superclass handle everything else */
        default:
            retval = (ULONG) DSM (cl, o, msg);
            break;
    }

    return (retval);
}
```

Boopsi *Rootclass* Methods

Since all classes should be subclasses of some class, with the exception of *rootclass*, all classes you write will be subclasses--perhaps indirectly so--of *rootclass*. Because of this, your class must either implement the *rootclass* methods or defer processing of these methods to the superclass (as DispatchmyTextLabel() did). Provided below are brief descriptions of the *rootclass* methods. Remember that any message unrecognized by a class dispatcher should be passed to the superclass (using the *amiga.lib* functions DSM() or DoSuperMethod()).

The *rootclass* method IDs that a subclass of *imageclass* needs to understand are:

OM_NEW	Create a new object.
OM_DISPOSE	Delete an object.
OM_SET	Change an object's attributes.
OM_GET	Retrieve the value of one of the object's attributes.

The dispatcher should pass other *rootclass* methods on to the superclass.

Each method requires one or more parameters. The MethodID is the only common parameter for each method.

OM_NEW

The OM_NEW method receives the following arguments:

```
struct opSet
{
    ULONG MethodID;
    struct TagItem *ops_AttrList;
    struct GadgetInfo *ops_GInfo;
}
```

The ops_AttrList field contains a pointer to the TagItem array of attributes for the new object. The ops_GInfo field is always NULL for the OM_NEW method.

Unlike other methods, this method is not passed an object pointer (since the whole idea is to create an object). The pointer normally used to pass a boopsi object is instead used to pass the address of the object's "true class" (the class the object is an instance of). That way, all class dispatchers can tell if they are the "true class" of the object being created (as opposed to a superclass of the true class). Also, with this pointer, *rootclass* can determine what the instance data is for an object, and can allocate the right amount of memory for it.

For the OM_NEW method, the new class's dispatcher should do the following:

1) Pass the message along to the superclass. *All* classes do this as *rootclass* takes care of allocating memory for the new object. As the OM_NEW method works "top down" (from *rootclass* down through its subclasses to the true class), each class will in turn initialize its corresponding instance data. This all happens before the new class's dispatcher regains control. Eventually, the message comes back from the superclass with a newly allocated object (unless of course something failed and you receive a NULL pointer instead).

2) Obtain a pointer to the object's instance data for this class. Use the INST_DATA() macro (defined in *<intuition/classes.h>*). INST_DATA() takes two arguments, a pointer to your class and a pointer to the object.

```
void *INST_DATA(*Class, *Object);
```

3) Initialize your instance data. You may allocate additional memory buffers for your object, or even create other objects which are components to objects of your class.

4) Process your initial attribute list (from the opSet structure passed in the OM_NEW message). In particular, process all the attributes that can be set only at initialization time. After you deal with the "initialization only" attributes, apply the same attribute processing on these remaining attributes that you would apply to an OM_SET message.

5) Return the object to the caller.

OM_DISPOSE

The OM_DISPOSE method instructs the class to deallocate an object. This method receives no parameters.

For the OM_DISPOSE method, the new class's dispatcher should do the following:

1) Free any additional memory you allocated (memory allocated in step 3 from OM_NEW).

2) Dispose of any objects that you created as components of your object (component objects created in step 3 from OM_NEW).

3) Pass the message up to the superclass, which will eventually reach *rootclass*, which will free the memory allocated for the object.

The *mytextlabelclass* example at the end of this article does not allocate any extra resources when it creates an object. Because it does not have to release any resources, the *mytextlabelclass* dispatcher lets its superclass handle the OM_DISPOSE method. Eventually, some superclass of *mytextlabelclass* will deallocate all of the memory for the OM_DISPOSEd object.

OM_SET

This method is used to set an object's attributes. The Intuition function SetAttr() calls this method. It receives the following arguments:

```
struct opSet
{
    ULONG MethodID;
    struct TagItem *ops_AttrList;
    struct GadgetInfo *ops_GInfo;
}
```

For the OM_SET method, the new class's dispatcher should process the attributes your class recognizes and have the superclass process any unrecognized attributes. Note that a subclass dispatcher can directly process attributes it inherits from a superclass, rather than passing the message on to the superclass.

Note that *mytextlabelclass* treats the OM_UPDATE method exactly like the OM_SET method. This is OK because these two methods are functionally equivalent for *imageclass* classes.

OM_GET

Retrieve an object's attribute. This method receives the following parameters:

```
struct opGet
{
    ULONG MethodID;
    ULONG opg_AttrID;
    ULONG *opg_Storage;
}
```

If the new class recognizes the attribute, the new class should fill in opg_Storage's target with the attribute's value. If the attribute is actually the attribute of some component object, you might want to pass the message on and let the component object process the OM_GET. If completely unrecognized, you should pass the message to your superclass.

Imageclass Methods

Imageclass defines several methods of its own which subclasses of imageclass either have to implement or pass on to their superclass. The method IDs for *imageclass* are defined in *<intuition/imageclass.h>*. Each method requires some parameters. The MethodID is the only parameter common to each method.

IM_DRAW

The IM_DRAW method is used to tell the image to render itself. The Intuition function DrawImageState() uses this method. IM_DRAW receives the following parameters:

```
struct impDraw
{
    ULONG MethodID;
    struct RastPort *imp_RPort;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Offset;

    ULONG imp_State;
    struct DrawInfo *imp_DrInfo;
}
```

The imp_State field contains the visual state to render the image. The visual states (defined in *<intuition/imageclass.h>*) are:

IDS_NORMAL	idle state
IDS_SELECTED	for selected gadgets.
IDS_DISABLED	for disabled gadgets.
IDS_BUSY	for future functionality
IDS_INDETERMINATE	for future functionality
IDS_INACTIVENORMAL	normal, in inactive window border.
IDS_INACTIVESELECTED	selected, in inactive border.
IDS_INACTIVEDISABLED	disabled, in inactive border.

When setting the pens to render an image, use the values from the imp_DrInfo->dri_Pens pen array (Note that it is possible that imp_DrInfo will be NULL). The possible pen values are defined in *<intuition/screens.h>*.

The following code fragment shows how to use the shadow color for rendering.

```
UWORD *pens = (imp->imp_DrInfo) ? imp->imp_DrInfo->dri_Pens : NULL;

if (pens)
{
    SetAPen (imp->imp_RPort, pens[SHADOWPEN]);
}
```

IM_ERASE

The IM_ERASE method tells an image to erase itself. The Intuition function EraseImage() uses this method. IM_ERASE receives the following parameters:

```
struct impErase
{
    ULONG MethodID;
    struct RastPort *imp_RPort;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Offset;
}
```

The *mytextlabelclass* example doesn't know anything about this method, so it blindly passes this message on to the superclass. The superclass, *imageclass*, will call the *graphics.library* function EraseRect() using the dimensions found in the *imageclass* object's Image structure.

IM_HITTEST

IM_HITTEST returns true if a point is within the image. The Intuition function PointInImage() uses this method. IM_HITTEST requires the following parameters:

```
struct impHitTest
{
    ULONG MethodID;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Point;
}
```

The *mytextlabelclass* example blindly passes this method on to its superclass. The superclass, *imageclass*, will return TRUE if the point is within the old Image structure box.

IM_DRAWFRAME

The IM_DRAWFRAME method instructs the image to render itself within the confines of the given rectangle. It receives the following parameters:

```
struct impDraw
{
    ULONG MethodID;
    struct RastPort *imp_RPort;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Offset;

    ULONG imp_State;
    struct DrawInfo *imp_DrInfo;

    struct
    {
        WORD Width;
        WORD Height;
    }imp_Dimensions;
}
```

The Width and Height fields provide the object's rectangular bounds. How the image object deals with the frame is implementation specific. Typically, a scaleable image will scale itself as best it can to fit into the rectangle. The *mytextlabelclass.c* example does not actually implement this method, instead *mytextlabelclass* treats IM_DRAWFRAME like the IM_DRAW method.

In general, applications that use this method to draw an object should use the IM_ERASEFRAME method (see below) to erase it. This will ensure that the image was erased at the proper scale.

IM_ERASEFRAME

The IM_ERASEFRAME method instructs an image confined to a given rectangle to erase itself. Normally this method is used to erase an image drawn using the IM_DRAWFRAME method. This method expects the following parameters:

```
struct impErase
{
    ULONG MethodID;
    struct RastPort *imp_RPort;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Offset;

    /* these parameters only valid for IM_ERASEFRAME */
    struct
    {
        WORD Width;
        WORD Height;
    }imp_Dimensions;
}
```

The *mytextlabelclass* example blindly passes this method on to its superclass. The superclass treats IM_ERASEFRAME just like IM_ERASE.

IM_HITFRAME

The IM_HITFRAME method is used to determine if a point is within an image that is contained within (or scaled to) the given rectangle. This method is intended to test images that were rendered using IM_DRAWFRAME. This method receives the following parameters:

```
struct impHitTest
{
    ULONG MethodID;
    struct
    {
        WORD X;
        WORD Y;
    }imp_Point;

    struct
    {
        WORD Width;
        WORD Height;
    }imp_Dimensions;
}
```

The *mytextlabelclass* example blindly passes this method on to its superclass. The superclass treat this meothd just like the IM_HITTEST method.

IM_MOVE

The IM_MOVE method erases and redraws an image. It is intended for use in some subclass of imageclass that performs animation or smooth dragging. Currently, no public boopsi classes implement this method.

IM_FRAMEBOX

The IM_FRAMEBOX method returns size information for an image (usually some sort of frame image). The following parameters are associated with the IM_FRAMEBOX method.

```
struct impFrameBox
{
    ULONG MethodID;
    struct IBox *imp_ContentsBox;  /* Application supplied IBox for the result */
    struct IBox *imp_FrameBox;     /* Rectangle to frame */
    struct DrawInfo *imp_DrInfo;
    ULONG imp_FrameFlags;
}
```

This method is used to ask the image what size it would like to be, if it had to frame the rectangle in the imp_FrameBox field. This method normally applies only to image classes that put a frame around some object (like *frameiclass*). By passing the dimensions and position of a rectangle, the framing image determines the position and size it should be to properly "frame" the object bounded by the imp_FrameBox rectangle. IM_FRAMEBOX stores the result in the IBox structure pointed to by imp_ContentsBox. This method allows an application to use a framing image without having to worry about image specific details such as accounting for the thickness of the frame or centering the frame around the object.

The imp_FrameFlags field is a bit field used to specify certain options for the IM_FRAMEBOX method. Currently, there is only one defined for it, FRAMEF_SPECIFY. If this bit is set, IM_FRAMEBOX has to use the width and height supplied to it in the imp_FrameBox field (even if these are too small!) as the frame dimensions. It can only adjust its position, typically to center the object as best as possible.

This method is not supported by the *mytextlabelclass* example. It passes this message to its superclass which does not support this method either. When the message returns from the superclass, the return value will be zero, indicating to the application that this method is not supported.

Image Class Example

The image class example code, *mytextlabelclass.c*, illustrates a complete custom image class. This image class provides an application with textual labels that have a particular character underlined. This is useful for indicating which key controls a gadget (although the example provided only utilizes *imageclass* objects; there are no gadgets involved).

A custom image can be used in the place of any standard Intuition Image structure. For example, an application can attach an *imageclass* object to: the GadgetRender and SelectRender fields of a Gadget structure (defined in *<intuition/intuition.h>*), the ReqImage field of a Requester structure, or even the ItemFill field of the MenuItem structure.

Under Intuition V36, the DrawImage() function passed an invalid DrawInfo structure, therefore it wasn't possible to use a custom *imageclass* object and the DrawImage() together. With V37, a NULL DrawInfo is passed when no valid DrawInfo is available.

The example code (*usemyIC.c*) initializes and uses a custom *imageclass* object. Notice that *usemyIC.c* directly manipulates fields within the Image structure embedded within the boopsi *imageclass* object. This is legal for image classes whose immediate superclass is *imageclass* (for the LeftEdge, TopEdge, Width, Height, ImageData, PlanePick, and PlaneOnOff Image structure fields only; the other Image structure fields are *off limits*). Indirect subclasses of *imageclass* may *not* alter the values in the embedded Image structure as future direct subclasses of *imageclass* may need to know about changes to values in the Image structure.

```
/*
 * texticlass.c Copyright (C) 1991 Commodore-Amiga, Inc. All Rights Reserved Worldwide
 * Written by David N. Junod
 *
 * Compiled with SAS/C 5.10a LC -cfist -ms -v (must be linked with classface.o and
 * hookface.o)
 *
 * The Image structure as used by this class:
 *
 * struct Image {
 *
 * SHORT    LeftEdge;        <----Offset relative to the container
 * SHORT    TopEdge;
 *
 * SHORT    Width;           <----Contains the text extent of the string
 * SHORT    Height;
 *
 * SHORT    Depth;           <----Maintained by boopsi (must be set to CUSTOMIMAGEDEPTH).
 *
 * USHORT   *ImageData;      <----Pointer to a NULL terminated text string
 *
 * UBYTE    PlanePick;       <----We use this for the foreground color
 *
 * UBYTE    PlaneOnOff;      <----We use this for the background color
 *
 * struct Image *NextImage;  <----Pointer to the next image.  Handled by DrawImage(). };
 */

#include <exec/types.h>
#include <exec/memory.h>
#include <exec/libraries.h>
#include <intuition/intuition.h>
#include <intuition/classes.h>
#include <intuition/classusr.h>
#include <intuition/cghooks.h>
#include <intuition/gadgetclass.h>
#include <intuition/imageclass.h>
#include <intuition/icclass.h>
#include <intuition/screens.h>
#include <graphics/gfx.h>
#include <graphics/gfxmacros.h>
#include <libraries/gadtools.h>
#include <utility/tagitem.h>
#include <clib/macros.h>
#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/utility_protos.h>
#include <string.h>

extern struct Library *SysBase, *DOSBase;
extern struct Library *IntuitionBase, *GfxBase, *UtilityBase;

/*
 * Because we are dealing with imageclass objects, the data structure that makes up the
 * object is an intuition Image structure.
 */
#define IM(o)    ((struct Image *)(o))

#define MYCLASSID       NULL
#define SUPERCLASSID    (IMAGECLASS)

Class          *initmyTextLabelClass(VOID);
ULONG           freemyTextLabelClass(Class * cl);
ULONG __saveds  dispatchmyTextLabel(Class * cl, Object * o, Msg msg);
ULONG           setmyTextLabelAttrs(Class * cl, Object * o, struct opSet * msg);
ULONG           getmyTextLabelAttr(Class * cl, Object * o, struct opGet * msg);
ULONG           drawmyTextLabel(Class * cl, Object * o, struct impDraw * msg);
WORD            aTextExtent(struct RastPort *, STRPTR, LONG, struct TextExtent *);
UWORD           GetLabelKeystroke(STRPTR label);
static VOID     getContentsExtent(Class * cl, Object * o, struct DrawInfo * drinfo);

/* prototypes of functions from classface.o */
ULONG           DoMethod(Object * o, ULONG methodID,...);
ULONG           DoSuperMethod(Class * cl, Object * o, ULONG methodID,...);
ULONG           CoerceMethod(Class * cl, Object * o, ULONG methodID,...);
ULONG           CM(Class * cl, Object * o, Msg msg);
ULONG           DM(Object * o, Msg msg);
ULONG           DSM(Class * cl, Object * o, Msg msg);
ULONG           SetSuperAttrs(Class * cl, Object * o, ULONG data,...);
```

```
struct localObjData
{
    /* Font to use */
    struct TextFont *lod_Font;

    /* The key that is underlined */
    UWORD           lod_Key;

    /* DrawMode */
    UBYTE           lod_Mode;
};


Class          *
initmyTextLabelClass(VOID)
{
    ULONG           hookEntry(); /* from hookface.o */
    Class          *cl;

    if (cl = MakeClass(MYCLASSID,
                       SUPERCLASSID, NULL,
                       sizeof(struct localObjData), 0))
    {
        /* Fill in the callback hook */
        cl->cl_Dispatcher.h_Entry = hookEntry;
        cl->cl_Dispatcher.h_SubEntry = dispatchmyTextLabel;
    }
    /* Return a pointer to the class */
    return (cl);
}


ULONG
freemyTextLabelClass(Class * cl)
{

    /* Try to free the public class */
    return ((ULONG) FreeClass(cl));
}


/*
 * The SAS "__saveds" flag tells the SAS compiler to put
 * the data storage address + 32766 into A4.
 */
ULONG           __saveds
dispatchmyTextLabel(Class * cl, Object * o, Msg msg)
{
    struct localObjData *lod;
    Object         *newobj;
    ULONG           retval;

    switch (msg->MethodID)
    {
    case OM_NEW:
        /* Pass up to the superclass... */
        if (newobj = (Object *) DSM(cl, o, msg))
        {
            struct TagItem *attrs = ((struct opSet *) msg)->ops_AttrList;
            struct DrawInfo *drinfo;

            /* Get the DrawInfo */
            drinfo = (struct DrawInfo *) GetTagData(SYSIA_DrawInfo, NULL, attrs);

            /* Get the instance data */
            lod = INST_DATA(cl, newobj);

            /* Establish defaults */
            IM(newobj)->PlanePick = 1;
            lod->lod_Mode = JAM1;

            /* Set the attributes */
            setmyTextLabelAttrs(cl, newobj, (struct opSet *) msg);

            /* Get the bounding rectangle of the label */
            getContentsExtent(cl, newobj, drinfo);
        }
        retval = (ULONG) newobj;
        break;
```

```
    case OM_GET:
        retval = getmyTextLabelAttr(cl, o, (struct opGet *) msg);
        break;

    case OM_UPDATE:
    case OM_SET:
        /* Do the superclass first */
        retval = DSM(cl, o, msg);

        /* Call our set routines */
        retval += setmyTextLabelAttrs(cl, o, (struct opSet *) msg);
        break;

    case IM_DRAW:               /* draw the label */
    case IM_DRAWFRAME:          /* drawmyTextLabel() will take care of
                                   extra framing info */
        retval = drawmyTextLabel(cl, o, (struct impDraw *) msg);
        break;

    /* Let the superclass handle everything else */
    default:
        retval = (ULONG) DSM(cl, o, msg);
        break;
    }

    return (retval);
}


/* Set attributes of an object */
ULONG
setmyTextLabelAttrs(Class * cl, Object * o, struct opSet * msg)
{
    struct localObjData *lod = INST_DATA(cl, o);
    struct TagItem *tags = msg->ops_AttrList;
    struct TagItem *tstate;
    struct TagItem *tag;
    ULONG           tidata;

    /* process rest */
    tstate = tags;
    while (tag = NextTagItem(&tstate))
    {
        tidata = tag->ti_Data;
        switch (tag->ti_Tag)
        {
        case IA_FGPen:
            IM(o)->PlanePick = (UBYTE) tidata;
            break;

        case IA_BGPen:
            IM(o)->PlaneOnOff = (UBYTE) tidata;
            break;

            /* Must be a TextFont pointer. */
        case IA_Font:
            /* Set the font */
            lod->lod_Font = (struct TextFont *) tidata;
            break;

            /* Drawing mode to use */
        case IA_Mode:
            lod->lod_Mode = (UBYTE) tidata;
            break;

        case IA_Data:
            IM(o)->ImageData = (USHORT *) tidata;
            lod->lod_Key = GetLabelKeystroke((STRPTR) tidata);
            break;
        }
    }

    return (1L);
}
```

```
/* Inquire attributes of an object */
ULONG
getmyTextLabelAttr(Class * cl, Object * o, struct opGet * msg)
{
    struct localObjData *lod = INST_DATA(cl, o);

    switch (msg->opg_AttrID)
    {
    case IA_Font:
        *msg->opg_Storage = (ULONG) lod->lod_Font;
        break;

    case IA_Mode:
        *msg->opg_Storage = (ULONG) lod->lod_Mode;
        break;

        /* Let the superclass try */
    default:
        return ((ULONG) DSM(cl, o, msg));
    }

    return (1L);
}


ULONG
drawmyTextLabel(Class * cl, Object * o, struct impDraw * msg)
{
    struct localObjData *lod = INST_DATA(cl, o);
    STRPTR          label = (STRPTR) IM(o)->ImageData;
    struct DrawInfo *di = msg->imp_DrInfo;
    struct RastPort *rp = msg->imp_RPort;
    struct TextFont *tf = NULL;
    WORD            len = strlen(label);
    WORD            left, top;
    WORD            height = 0;
    WORD            width = 0;
    WORD            i;

    /* Clear the key */
    lod->lod_Key = NULL;

    /* Get a pointer to the font to use */
    if (!(tf = lod->lod_Font) && di)
    {
        tf = di->dri_Font;
    }

    /* Make sure we have font pointer */
    if (tf)
    {
        /* Set the font */
        SetFont(rp, tf);
    }
    /* Figure out our coordinates */
    top = msg->imp_Offset.Y + IM(o)->TopEdge + rp->TxBaseline;
    left = msg->imp_Offset.X + IM(o)->LeftEdge;

    /* See if we have frame information. */
    if (msg->MethodID == IM_DRAWFRAME)
    {
        /* Center the text inside the frame. */
        width = msg->imp_Dimensions.Width;
        height = msg->imp_Dimensions.Height;
        top += ((height - IM(o)->Height) > 0) ? ((height - IM(o)->Height) / 2) : 0;
        left += ((width - IM(o)->Width) > 0) ? ((width - IM(o)->Width) / 2) : 0;
    }

    /* Set the colors */
    SetAPen(rp, IM(o)->PlanePick);
    SetBPen(rp, IM(o)->PlaneOnOff);

    /* Set the drawing mode */
    SetDrMd(rp, lod->lod_Mode);

    /* Move to the start */
    Move(rp, left, top);
```

```
    /* Step through string */
    for (i = 0; i < (len - 1); i++)
    {
        /* Is this an '_' ? */
        if (label[i] == '_')
        {
            WORD            bot = (top + rp->TxHeight - rp->TxBaseline);
            WORD            mark;

            /* Draw the first part of the string */
            Text(rp, label, i);

            /* Remember where we are in the string */
            mark = rp->cp_x;

            /* Draw the underscore */
            Move(rp, mark, bot);
            Draw(rp, (mark + TextLength(rp, &label[(i + 1)], 1L) - 2), bot);

            /* Return to where we were */
            Move(rp, mark, top);

            /*
             * Draw the rest of the string.  This one is done last so that the cursor
             * could be positioned after the text.
             */
            Text(rp, &label[(i + 1)], (len - i - 1));

            /* Return the underlined character */
            lod->lod_Key = (UWORD) label[i];
        }
    }

    /* Do we have an underscore? */
    if (!lod->lod_Key)
    {
        /* Didn't find an '_' sign */
        Text(rp, label, len);
    }
    return (1L);
}


UWORD
GetLabelKeystroke(STRPTR label)
{
    LONG            count = (label) ? strlen(label) : 0L;
    LONG            i;

    /* Search for an _ sign */
    for (i = 0; i < (count - 1); i++)
    {
        /* Did we find an _ sign? */
        if (label[i] == '_')
        {
            return ((UWORD) label[(i + 1)]);
        }
    }

    return (0);
}


/* TextExtent that honors the '_' as being a non-printable character (once) */
WORD
aTextExtent(struct RastPort * rp, STRPTR string, LONG count, struct TextExtent * te)
{
    WORD retval = FALSE;
    STRPTR buffer;
    LONG i;

    /* Allocate a temporary buffer */
    if (buffer = AllocVec ((count + 1), MEMF_CLEAR))
    {
       /* Step through string */
       for (i = 0; i < count; i++)
       {
            /* Is this an '_' sign? */
            if (string[i] == '_')
            {
              /* Add the rest of the label to the buffer */
              strcat (buffer, &string[(i + 1)]);

              /* Adjust the length of the string. */
              count--;
              break;
            }
            else
            {
              /* Copy each character over, until we reach the _ mark */
              buffer[i] = string[i];
            }
       }

       /* Get the extent */
       TextExtent(rp, buffer, count, te);

      /* Free the temporary buffer */
      FreeVec (buffer);

      /* Show that we were successful */
      retval = TRUE;
    }

    /* Return whatever textextent returned */
    return (retval);
}

static          VOID
getContentsExtent(Class * cl, Object * o, struct DrawInfo * drinfo)
{
    struct localObjData *lod = INST_DATA(cl, o);
    struct TextExtent te =
    {NULL};
    struct RastPort rp;
    STRPTR          label;

    /* maybe look at some flags to handle other types of text someday */
    if (label = (STRPTR) IM(o)->ImageData)
    {
        /* Initialize the RastPort */
        InitRastPort(&rp);

        if (lod->lod_Font)
        {
            SetFont(&rp, lod->lod_Font);
        }
        else if (drinfo && drinfo->dri_Font)
        {
            SetFont(&rp, drinfo->dri_Font);
        }
        /* Get the rectangle for the label */
        aTextExtent(&rp, label, strlen(label), &te);

        /* Set the image structure */
        IM(o)->Width = te.te_Width;
        IM(o)->Height = te.te_Height;
    }
    else
    {
        IM(o)->Width = IM(o)->Height = 0;
    }
}
```

```
/*
 * example.c Copyright (C) 1991 Commodore-Amiga, Inc. All Rights Reserved Worldwide
 * Written by David N. Junod
 *
 * Compiled with SAS/C 5.10a LC -cfist -ms -v (must be linked with
 * texticlass.o, classface.o and hookface.o)
 *
 */

#include <exec/types.h>
#include <exec/libraries.h>
#include <intuition/intuition.h>
#include <intuition/classes.h>
#include <intuition/classusr.h>
#include <intuition/cghooks.h>
#include <intuition/gadgetclass.h>
#include <intuition/imageclass.h>
#include <graphics/gfx.h>
#include <graphics/gfxmacros.h>
#include <libraries/gadtools.h>
#include <utility/tagitem.h>
#include <clib/macros.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/utility_protos.h>
#include <string.h>

extern struct Library *SysBase, *DOSBase;
struct Library *IntuitionBase, *GfxBase, *UtilityBase;

Class          *initmyTextLabelClass(VOID);
ULONG           freemyTextLabelClass(Class * cl);

VOID
main(VOID)
{
    Class          *cl;
    struct Image   *im;
    struct Window  *win;
    struct RastPort *rp;
    UWORD           top, left, height;

    /* Make sure we're at least using Version 2.0 */
    if (IntuitionBase = OpenLibrary("intuition.library", 36))
    {
        GfxBase = OpenLibrary("graphics.library", 36);
        UtilityBase = OpenLibrary("utility.library", 36);

        /* Open a window, without system gadgets or IDCMP events */
        if (win = OpenWindowTags(NULL,
                                 WA_Left, 10,
                                 WA_Top, 10,
                                 WA_Width, 320,
                                 WA_Height, 100,
                                 TAG_DONE))
        {
            /* Cache the pointer to the RastPort */
            rp = win->RPort;

            /* Cache the upper-left coordinates of the window */
            top = win->BorderTop + INTERHEIGHT;
            left = win->BorderRight + INTERWIDTH;

            /* Cache the height of the font */
            height = rp->TxHeight + INTERHEIGHT;

            /* Initialize the custom image class. */
            if (cl = initmyTextLabelClass())
            {
                /* Create a new image structure, using the given string. */
                if (im = NewObject(cl, NULL,
                                   IA_Data, (ULONG) "Line _1",
                                   TAG_DONE))
                {
                    /* Paint using the provided text string. */
                    DrawImageState(rp, im, left, top,
                                   IDS_NORMAL, NULL);

                    /* Replace the text string, and paint it. */
                    im->ImageData = (USHORT *) "Line _2";
                    DrawImageState(rp, im, left, top + height,
                                   IDS_NORMAL, NULL);

                    /* Replace the text string, and paint it. */
                    im->ImageData = (USHORT *) "Line _3";
                    DrawImageState(rp, im, left, top + (height * 2),
                                   IDS_NORMAL, NULL);

                    /* Free the image. */
                    DisposeObject(im);
                }

                /* Free the image class. */
                freemyTextLabelClass(cl);
            }

            Delay(250);
            CloseWindow(win);
        }

        CloseLibrary(UtilityBase);
        CloseLibrary(GfxBase);
        CloseLibrary(IntuitionBase);
    }
}
```

❖

Amiga Mail
Volume II
May/June 1991

Boopsi's Transparent Base Classes

by Peter Cherna

[Editor's note: This article assumes the reader is already familar with several boopsi concepts. For more information on boopsi, see the article "Introduction to Boopsi" from the March/April 1991 issue of Amiga Mail.]

Although it is normally considered a "programming sin", in some cases it is legal to directly manipulate some internal fields of certain boopsi objects. For compatibility reasons, a boopsi Image or Gadget object contains an actual Image or Gadget structure. To a large degree, a boopsi Image or Gadget looks like a regular Image or Gadget. Most fields in these embedded Image and Gadget structures generally contain information equivalent to the information in a conventional Image or Gadget (although not always). This means that many programs and routines that don't know about boopsi can still make some use of boopsi Images and Gadgets. These objects are instances of the *Transparent Base Classes, imageclass* and *gadgetclass.*

To change an attribute of a boopsi object, you normally invoke a set method (`OM_SET`). The Intuition functions SetAttrs() and SetGadgetAttrs() invoke this method. A boopsi class is informed of any attribute change at that time, allowing it to react to this change. The reaction can include validation of the changed attribute, changing other attributes to match, or informing other objects of the change. That is the inherent advantage of a function-access approach to changing attributes.

When using conventional Images and Gadgets, you generally modify the structure's fields directly. This operation is very fast. For such Images and Gadgets, there is no class that needs to know about the changes, so there is no problem. However, this is untrue of *boopsi* Images and Gadgets. Although directly modifying the boopsi object's internal structure would provide a performance increase over using the boopsi `OM_SET` mechanism, altering a boopsi object's internal structure directly will not give the class the opportunity to react to any structure changes. This violates the boopsi concept, and therefore cannot be done in general.

In order to provide a balance between the flexibility of function access and the performance of direct access, the transparent base classes *imageclass* and *gadgetclass* will not depend on being informed of changes to certain fields in the internal Image and Gadget structures. This means that direct subclasses of *imageclass* or *gadgetclass* (for example, *frameiclass*, *sysiclass*, *propgclass*, *buttongclass*, or any direct subclass you create) are allowed to poke specific fields. Indirect subclasses of *imageclass* or *gadgetclass* (for example, subclasses of *propgclass*, *frameiclass*, or other indirect subclasses you create) may not poke those fields, since their parent classes may depend on hearing about changes to these fields, which the SetAttrs() call (or similar function) provides.

For images, the following Image structure fields are the only fields that may be poked by image classes whose immediate superclass is *imageclass*:

LeftEdge
TopEdge
Width
Height
ImageData
PlanePick
PlaneOnOff

For gadgets, the following Gadget structure fields are the only fields that may be poked by gadget classes whose immediate superclass is *gadgetclass*:

LeftEdge
TopEdge
Width
Height
Flags
Activation
GadgetType
GadgetRender
SelectRender
GadgetText
SpecialInfo

In all cases, the direct subclass must take care to maintain sensible values for these fields.

Under ***no*** circumstances may an indirect subclass poke one of these fields, ***even if*** the subclass knows the superclasses do not depend on notification for this field. This is the only way to preserve the possibility for future enhancements to that superclass.

❖

Amiga Mail
Volume II
May/June 1991

From the Desk of Jeff Scherb

I was looking through some recent issues of Amiga Magazines and was reminded how far the Amiga has come in both professional productivity software and professional multimedia.

Flipping quickly through these magazines, I noticed several things. The Amiga now has an industrial strength database, compatible with databases on the PC (Precision Inc.'s *Superbase Professional 4*). There are also several high quality structured drawing packages—the new kid on the block is *Designworks*, from New Horizons. There is an excellent presentation graphics package, *Scala*, from GVP and there are also three professional desktop publishing packages (*PageStream*, from Soft Logik, and *Saxon Publisher*, from Saxon, *Professional Page*, from Gold Disk).

OK, so we've got databases, structured drawing packages, business presentation software, and desktop publishing software. Each of the Amiga products mentioned above (and many that I did not mention) competes favorably with Mac and PC products. About all the Amiga needs now is a spreadsheet that can hold its own against the PC competition. I'm not talking about a spreadsheet that can win a head-to-head feature war with Excel or 123/G—no, I'd like to see a solid GUI spreadsheet that I could do the CATS budget with. Throw in PostScript and multiple font support, some simple drawing and annotation features, and perhaps an outlining feature, and I'll be happy.

One area where the Amiga really shines is in professional multimedia. The same magazines told me about the Video Toaster and a wide range of other gear for the professional video studio. Digital Processing Systems, Inc. now has a time base corrector that plugs into an AT-type slot; Digital Creations has a TBC called the *Kitchen Sync*; and RGB has a very impressive edit controller that interfaces through *ARexx* to the Toaster. The Amiga has a whole slew of new 24-bit cards and software should be available soon that can take advantage of all those bits. Personally, I can't wait to get my hands on one of those new 16-bit audio samplers from Sunrize.

All of the products mentioned above were found in two recent issues of Amiga magazines (disclaimer: if I left your product out, it's probably because I didn't notice it in those magazines). A year ago, we at Commodore were talking about the Amiga growing up. With all the professional applications and peripherals now in evidence, I think we can all be proud of "our child"—she really has grown up. Now, with any luck, she'll be on a "fast track" in the corporate world.

-JS

Table of Contents

May/June 1991

I
Programming Practices and Standards
general purpose Amiga programming

Tag Items and Tag Lists . . . . . I - 1
staff

2.0 Version Strings . . . . . I - 3
Carolyn Scheppner

The ASL Library. . . . . . I - 7
John Orr

II
AmigaDOS
dos.library

Executing External Programs with Release 2.0 . . . . . II - 1
Carolyn Scheppner

AmigaDOS Packet Interface Specification . . . . . II - 5
John Toebes

Standard Command Line Parsing . . . . . II - 27
John Orr

III
Exec
the multitasking executive, exec.library

Using SetFunction() in a Debugger . . . . . III - 1
Ewout Walraven

IV
Intuition and Workbench
the Amiga user interface

Obtaining an Icon Image . . . . . IV - 1
David Junod

An Introduction to V36 Screens and Windows . . . . . IV - 3
Ewout Walraven

Opening Screens and Windows on Any Amiga . . . . . IV - 17
Ewout Walraven

Introduction to Boopsi . . . . . IV - 23
John Orr

Writing a Boopsi Image Class . . . . . IV - 41
David N. Junod

Boopsi's Transparent Base Classes . . . . . IV - 57
Peter Cherna

V
Graphics
animation, text, the Blitter, layers.library

Using the ECS Genlocking Features . . . . . V - 1
Ewout Walraven

VI
Sound and Music
audio, speech, MIDI

VII
IFF
iffparse.library and other IFF topics

The DR2D FORM . . . . . VII - 1
Ross Cunniff and John Orr

VIII
File Systems, Handlers, and Storage Devices
DOS packet handlers, storage media, SCSI

IX
Timer, Serial, and Parallel
timer.device, serial port, parallel port

X
Output
printers, printing, Postscript

XI
Input
user input, Commodities Exchange

Introduction to Commodities Exchange . . . . . XI - 1
John Orr

```
ACTION_EXAMINE_OBJECT    23       Examine(...)
ARG1:   LOCK     Lock of object to examine
ARG2:   BPTR     FileInfoBlock to fill in

RES1:   BOOL     Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE     Failure code if RES1 = DOSFALSE
```

This action fills in the FileInfoBlock with information about the locked object. The Examine() function uses this packet. This packet is actually used for two different types of operations. It is called to obtain information about a given object while in other cases, it is called to prepare for a sequence of EXAMINE_NEXT operations in order to traverse a directory.

This seemingly simple operation is not without its quirks. One in particular is the FileInfoBlock->fib_Comment field. This field used to be 116 bytes long, but was changed to 80 bytes in release 1.2. The extra 36 bytes lie in the fib_Reserved field. Another quirk of this packet is that both the fib_EntryType and the fib_DirEntryType fields must be set to the same value, as some programs look at one field while other programs look at the other.

File systems should use the same values for fib_DirEntryType as the ROM file system and ram-handler do. These are as follows:

```
ST_ROOT          1
ST_USERDIR       2
ST_SOFTLINK      3 NOTE: this Shows up as a directory unless checked for explicitly
ST_LINKDIR       4
ST_FILE         -3
ST_LINKFILE     -4
```

Also note that for directories, handlers ***must*** use numbers greater than 0, since some programs test to see if fib_DirEntryType is greater than zero, ignoring the case where fib_DirEntryType equals 0. Handlers should avoid using 0 because it is not interpreted consistently.

```
ACTION_EXAMINE_NEXT      24       ExNext(...)
ARG1:   LOCK     Lock on directory being examined
ARG2:   BPTR     BPTR FileInfoBlock

RES1:   BOOL     Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE     Failure code if RES1 = DOSFALSE
```

The ExNext() function uses this packet to obtain information on all the objects in a directory. ACTION_EXAMINE fills in a FileInfoBlock structure describing the first file or directory stored in the directory referred to in the lock in ARG1. ACTION_EXAMINE_NEXT is used to find out about the rest of the files and directories stored in the ARG1 directory. ARG2 contains a pointer to a valid FileInfoBlock field that was filled in by either an ACTION_EXAMINE or a previous ACTION_EXAMINE_NEXT call. It uses this structure to find the next entry in the directory. This packets writes over the old FileInfoBlock with information on the next file or directory in the ARG2 directory. ACTION_EXAMINE_NEXT returns a failure code of ERROR_NO_MORE_ENTRIES when there are no more files or directories left to be examined. Unfortunately, like ACTION_EXAMINE, this packet has its own peculiarities. Among the quirks that ACTION_EXAMINE_NEXT must account for are:

• The situation where an application calls ACTION_EXAMINE_NEXT one or more times and then stops invoking it before encountering the end of the directory.

• The situation where a FileInfoBlock passed to ACTION_EXAMINE_NEXT is not the same as the one passed to ACTION_EXAMINE or even the previous EXAMINE_NEXT operation. Instead, it is a copy of the FileInfoBlock with only the fib_DiskKey and the first 30 bytes of the fib_FileName fields copied over. *This is now considered to be illegal and will not work in the future. Any new code should not be written in this manner.*

• Because a handler can receive other packet types between ACTION_EXAMINE_NEXT operations, the ACTION_EXAMINE_NEXT function must handle any special cases that may result.

• The LOCK passed to ACTION_EXAMINE_NEXT is not always the same lock used in previous operations. It is however a lock on the same object.

Because of these problems, ACTION_EXAMINE_NEXT is probably the trickiest action to write in any handler. Failure to handle any of the above cases can be quite disastrous.

```
ACTION_CREATE_DIR          22      CreateDir(...)
ARG1:    LOCK     Lock to which ARG2 is relative
ARG2:    BSTR     Name of new directory  (relative to ARG1)

RES1:    LOCK     Lock on new directory
RES2:    CODE     Failure code if RES1 = DOSFALSE

ACTION_DELETE_OBJECT       16      DeleteFile(...)
ARG1:    LOCK     Lock to which ARG2 is relative
ARG2:    BSTR     Name of object to delete (relative to ARG1)

RES1:    BOOL     Success/failure (DOSTRUE/DOSFALSE)
RES2:    CODE     Failure code if RES1 = DOSFALSE

ACTION_RENAME_OBJECT       17      Rename(...)
ARG1:    LOCK     Lock to which ARG2 is relative
ARG2:    BSTR     Name of object to rename (relative to ARG1)
ARG3:    LOCK     Lock associated with target directory
ARG4:    BSTR     Requested new name for the object

RES1:    BOOL     Success/failure (DOSTRUE/DOSFALSE)
RES2:    CODE     Failure code if RES1 = DOSFALSE
```

These three actions perform most of the work behind the AmigaDOS commands *MakeDir*, *Delete*, and *Rename* (for single files). These packets take as their parameters a lock describing where the file is and a name relative to that lock. It is the responsibility of the file system to ensure that the operation is not going to cause adverse effects. In particular, the RENAME_OBJECT action allows moving files across directory bounds and as such must ensure that it doesn't create hidden directory loops by renaming a directory into a child of itself.

For Directory objects, the DELETE_OBJECT action must ensure that the directory is empty before allowing the operation.

```
ACTION_PARENT              29      Parent(...)
ARG1:    LOCK     Lock on object to get the parent of

RES1:    LOCK     Parent Lock
RES2:    CODE     Failure code if RES1 = 0
```

This action receives a lock on an object and creates a shared lock on the object's parent. If the original object has no parent, then a lock of 0 is returned. Note that this operation is typically used in the process of constructing the absolute path name of a given object.

```
ACTION_SET_PROTECT        21      SetProtection(...)
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of object (relative to ARG2)
ARG4:   LONG    Mask of new protection bits

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to modify the protection bits of an object. The 4 lowest order bits (RWED) are a bit peculiar. If their respective bit is set, that operation is not allowed (i.e. if a file's delete bit is set the file is *not* deleteable). By default, files are created with the RWED bits set and all others cleared. Additionally, any action which modifies a file is required to clear the A (archive) bit. See the *dos/dos.h* include file for the definitions of the bit fields.

```
ACTION_SET_COMMENT        28      SetComment(...)
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of object (relative to ARG2)
ARG4:   BSTR    New Comment string

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to set the comment string of an object. If the object does not exist then DOSFALSE will be returned in RES1 with the failure code in RES2. The comment string is limited to 79 characters.

```
ACTION_SET_DATE           34      SetFileDate(...) in 2.0
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of Object (relative to ARG2)
ARG4:   CPTR    DateStamp

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to set an object's creation date.

2.0 only

```
ACTION_FH_FROM_LOCK       1026    OpenFromLock(lock)
ARG1:   BPTR    BPTR to file handle to fill in
ARG2:   LOCK    Lock of file to open

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = NULL
```

This action open a file from a given lock. If this action is successful, the file system will essentially steal the lock so a program should not use it anymore. If ACTION_FH_FROM_LOCK fails, the lock is still usable by an application.

2.0 only

```
ACTION_SAME_LOCK          40      SameLock(lock1,lock2)
ARG1:   BPTR    Lock 1 to compare
ARG2:   BPTR    Lock 2 to compare

RES1:   LONG    Result of comparison, one of
  LOCK_SAME           (0) if locks are for the same object
  LOCK_SAME_HANDLER   (1) if locks are on different objects of same handler
  LOCK_DIFFERENT     (-1) otherwise
RES2:   CODE    Failure code if RES1 is LOCK_DIFFERENT
```

This action compares the targets of two locks. If they point to the same object, ACTION_SAME_LOCK should return LOCK_SAME.

2.0 only

```
ACTION_MAKE_LINK          1021    MakeLink(name,targ,mode)
ARG1:   BPTR    Lock on directory ARG2 is relative to
ARG2:   BSTR    Name of the link to be created (relative to ARG1)
ARG3:   BPTR    Lock on target object or name (for soft links).
ARG4:   LONG    Mode of link, either LINK_SOFT or LINK_HARD

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This packet causes the file system to create a link to an already existing file or directory. There are two kinds of links, hard links and soft links. The basic difference between them is that a file system resolves a hard link itself, while the file system passes a string back to DOS telling it where to find a soft linked file or directory. To the packet level programmer, there is essentially no difference between referencing a file by its original name or by its hard link name. In the case of a hard link, ARG3 is a lock on the file or directory that the link is "linked" to, while in a soft link, ARG3 is a pointer (CPTR) to a C-style string.

In an over-simplified model of the ROM file system, when asked to locate a file, the system scans a disk looking for a file header with a specific (file) name. That file header points to the actual file data somewhere on the disk. With hard links, more than one file header can point to the same file data, so data can be referenced by more than one name. When the user tries to delete a hard link to a file, the system first checks to see if there are any other hard links to the file. If there are, only the hard link is deleted, the actual file data the hard link used to reference remains, so the existing hard links can still use it. In the case where the original link (not a hard or soft link) to a file is deleted, the file system will make one of its hard links the new "real" link to the file. Hard links can exist on directories as well. Because hard links "link" directly to the underlying media, hard links in one file system cannot reference objects in another file system.

Soft links are resolved through DOS calls. When the file system scans a disk for a file or directory name and finds that the name is a soft link, it returns an error code (ERROR_IS_SOFT_LINK). If this happens, the application must ask the file system to tell it what the link the link refers to by calling ACTION_READ_LINK. Soft Links are stored on the media, but instead of pointing directly to data on the disk, a soft link contains a path to its object. This path can be relative to the lock in ARG1, relative to the volume (where the string will be prepended by a colon ':'), or an absolute path. An absolute path contains the name of another volume, so a soft link can reference files and directories on other disks.

2.0 only

```
ACTION_READ_LINK          1024  ReadLink(port,lck,nam,buf,len)
ARG1:   BPTR    Lock on directory that ARG2 is relative to
ARG2:   CPTR    Path and name of link (relative to ARG1).  NOTE: This is a C
string not a BSTR
ARG3:   APTR    Buffer for new path string
ARG4:   LONG    Size of buffer in bytes

RES1:   LONG    Actual length of returned string, -2 if there isn't enough
space in buffer,or -1 for other errors
RES2:   CODE    Failure code
```

This action reads a link and returns a path name to the link's object. The link's name (plus any necessary path) is passed as a CPTR (ARG2) which points to a C-style string, *not a BSTR*. ACTION_READ_LINK returns the path name in ARG3. The length of the target string is returned in RES1 (or a -1 indicating an error).

2.0 only **ACTION_CHANGE_MODE** 1028 **ChangeMode(type,obj,mode)**

```
ARG1:   LONG    Type of object to change - either CHANGE_FH or CHANGE_LOCK
ARG2:   BPTR    object to be changed
ARG3:   LONG    New mode for object - see ACTION_FINDINPUT, and
ACTION_LOCATE_OBJECT

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This action requests that the handler change the mode of the given file handle or lock to the mode in ARG3. This request should fail if the handler can't change the mode as requested (for example an exclusive request for an object that has multiple users).

2.0 only **ACTION_COPY_DIR_FH** 1030 **DupLockFromFH(fh)**

```
ARG1:   LONG    fh_Arg1 of file handle

RES1:   BPTR    Lock associated with file handle or NULL
RES2:   CODE    Failure code if RES1 = NULL
```

This action requests that the handler return a lock associated with the currently opened file handle. The request may fail for any restriction imposed by the file system (for example when the file handle is not opened in a shared mode). The file handle is still usable after this call, unlike the lock in ACTION_FH_FROM_LOCK.

2.0 only **ACTION_PARENT_FH** 1031 **ParentOfFH(fh)**

```
ARG1:   LONG    fh_Arg1 of File handle to get parent of

RES1:   BPTR    Lock on parent of a file handle
RES2:   CODE    Failure code if RES1 = NULL
```

This action obtains a lock on the parent directory (or root of the volume if at the top level) for a currently opened file handle. The lock is returned as a shared lock and must be freed. Note that unlike ACTION_COPY_DIR_FH, the mode of the file handle is unimportant. For an open file, ACTION_PARENT_FH should return a lock under all circumstances.

2.0 only **ACTION_EXAMINE_ALL** 1033 **ExAll(lock,buff,size,type,ctl)**

```
ARG1:   BPTR    Lock on directory to examine
ARG2:   APTR    Buffer to store results
ARG3:   LONG    Length (in bytes) of buffer (ARG2)
ARG4:   LONG    Type of request - one of the following:
           ED_NAME Return only file names
           ED_TYPE Return above plus file type
           ED_SIZE Return above plus file size
           ED_PROTECTION Return above plus file protection
           ED_DATE Return above plus 3 longwords of date
           ED_COMMENT Return above plus comment or NULL
ARG5:   BPTR    Control structure to store state information.  The control
structure must be allocated with AllocDosObject()!

RES1:   LONG    Continuation flag - DOSFALSE indicates termination
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This action allows an application to obtain information on multiple directory entries. It is particularly useful for applications that need to obtain information on a large number of files and directories.

This action fills the buffer (ARG2) with partial or whole ExAllData structures. The size of the ExAllData structure depends on the type of request. If the request type field (ARG4) is set to ED_NAME, only the ed_Name field is filled in. Instead of copying the unused fields of the ExAllData structure into the buffer, ACTION_EXAMINE_ALL truncates the unused fields. This effect is cumulative, so requests to fill in other fields in the ExAllData structure causes all fields that appear in the structure *before* the requested field will be filled in as well. Like the ED_NAME case mentioned above, any field that appears after the requested field will be truncated (see the ExAllData structure below). For example, if the request field is set to ED_COMMENT, ACTION_EXAMINE_ALL fills in all the fields of the ExAllData structure, because the ed_Comment field is last. This is the only case where the packet returns entire ExAllData structures.

```
struct ExAllData {
        struct ExAllData *ed_Next;
        UBYTE  *ed_Name;
        LONG    ed_Type;
        ULONG   ed_Size;
        ULONG   ed_Prot;
        ULONG   ed_Days;
        ULONG   ed_Mins;
        ULONG   ed_Ticks;
        UBYTE  *ed_Comment;     /* strings will be after last used field */
};
```

Each ExAllData structure entry has an ead_Next field which points to the next ExAllData structure. Using these links, a program can easily chain through the ExAllData structures without having to worry about how large the structure is. *Do not examine the fields beyond those requested* as they certainly will not be initialized (and will probably overlay the next entry).

The most important part of this action is the ExAllControl structure. It ***must*** be allocated and freed through AllocDosObject()/FreeDosObject(). This allows the structure to grow if necessary with future revisions of the operating and file systems. Currently, ExAllControl contains four fields:

Entries - This field is maintained by the file system and indicates the actual number of entries present in the buffer after the action is complete. Note that a value of zero is possible here as no entries may match the match string.

LastKey - This field ***must*** be initialized to 0 by the calling application before using this packet for the first time. This field is maintained by the file system as a state indicator of the current place in the list of entries to be examined. The file system may test this field to determine if this is the first or a subsequent call to this action.

MatchString - This field points to a pattern matching string to control which directory entries are returned. If this field is NULL, then all entries are returned. Otherwise, this string is used to pattern match the names of all directory entries before putting them into the buffer. The default AmigaDOS pattern match routine is used unless MatchFunc is not NULL (see below). Note that it is not acceptable for the application to change this field between subsequent calls to this action for the same directory.

MatchFunc - This field contains a pointer to an alternate pattern matching routine to validate entries. If it is NULL then the standard AmigaDOS wild card routines will be used. Otherwise, MatchFunc points to a hook function that is called in the following manner: